AN ABSTRACT OF THE THESIS OF

Arshad Alam for the degree of Master of Science in Mechanical Engineering presented on March 05 , 1992.

TITLE : Robustness Estimation via Integral Liapunov Functions

Redacted for Privacy

ABSTRACT APPROVED:

Andrzej Olas

An investigation focusing on methods of estimation of robustness of nominally linear dynamic systems with unstructured uncertainties was performed.

The algorithm proposed involves the consideration of an associated system, selection, and subsequent development, of Liapunov function candidate and integration of their derivatives along the solution trajectory.

A nominally linear multi-dimensional dynamic system is considered with unstructured, nonlinear, time-varying and bounded perturbations. The examples illustrate the success of the method: better estimates of the bounds, than those which results from traditional approaches were obtained.

Robustness of linear, time-invariant systems

subject to nonlinear, time-varying perturbations has been a matter of considerable research interest recently. Design of conventional state-feedback controllers requires knowledge of the bounds for disturbances. The knowledge of disturbance bounds is also important in adaptive control and control of nonlinear & uncertain systems. Numerous applications can be found in the fields of automation, aircraft control, manipulator trajectory control, etc.

The technique for the determination of robust stability bounds proposed in this paper can be utilized effectively in computerized robust control system design.

Robustness Estimation via Integral Liapunov Functions

by

Arshad Alam

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of
Master of Science

Completed March 05 , 1992
Commencement June 1992

APPROVED:

Redacted for Privacy

Assistant Professor of Mechanical Engineering in charge of major

Redacted for Privacy

Head of the Department of Mechanical Engineering

Redacted for Privacy

Dean of the Graduate School

Date thesis is presented March 05 , 1992 .
Typed by : Arshad Alam .

ACKNOWLEDGEMENT

I would like to thank my major Professor , Dr.A.Olas , who guided me through this work . Thanks are also due to Dr.C.E.Smith , my minor Professor , from whom I took most of my courses . My professors never got impatient answering my questions (including the stupid ones) .

I wish to express my sincerest gratitude to Dr.G.M.Reistad for providing me with funds . Thanks to all of my committee members and professors . Thanks to Dr. Robert Paasch and UCS for letting me use kepler and FPS .

I wish to acknowledge the help I had always received from Barb and Carb , my host family in Corvallis, since the day I arrived at Portland about two years ago .

Finally , I would like to thank all of my family members for their encouragement , wishes and financial support.

TABLE OF CONTENTS

1. INTRODUCTION 1

1.1 Design Objectives 1
1.2 Motivation 2
1.3 Engineering Applications 4

2. DEFINITIONS AND CONCEPTS 7

2.1 Stability In The Sense Of Liapunov 7
2.2 Liapunov's Direct Method 12

3. LIAPUNOV FUNCTION CANDIDATE 15

3.1 Integral Liapunov Function 15

4. ALGORITHM AND EXAMPLES 17

4.1 Introduction 17
4.2 Algorithm 18
4.3 Example One 22
4.4 Example Two 25

5. RESULTS AND CONCLUSIONS 27

5.1 Result Of Example One 27
5.2 Result Of Example Two 27
5.3 Conclusions 28
5.4 Future Study 28

BIBLIOGRAPHY 30

APPENDICES 32

APPENDIX I Equations for example one 32
APPENDIX II Equations for example two 35
APPENDIX III Source Codes 36

Robustness Estimation via Integral Liapunov Functions

1. INTRODUCTION

1.1 Design Objectives :

This paper presents an investigation on the "conservatism" in the time domain stability robustness bounds obtained by the Liapunov approach . The system considered is nominally stable and subject to nonlinear , time-varying , unstructured perturbation.

We describe the system as $\dot{x} = A_N x + e$ where $x \in R^n$ and $e = e(t,x)$, $e(t,0) = 0$, is an unstructured perturbation . We define the problem as the assignment to determine the largest number $k > 0$ such that if $\|e\| < k\|x\|$ then the trivial solution to the aforementioned system is stable in the sense of Liapunov.

The Liapunov direct method had been utilized in a number of applications for control system analysis and design during the past few years . Siljak in [14] and Patel et al. in [15] first proposed the procedures to calculate quantitative measures of robustness for both structured and unstructured perturbations . Yedavalli and Liang in [13] used transformations to obtain improvements in bounds estimaton . In the same paper , they also proposed an algorithm to find the "optimal" transformation for the case of a diagonal transformation . Results of [15] and [16] are based on the selection of some quadratic form as the Liapunov function . Becker and Grimm in [2] proved that application of

the small gain theorem provides the stability bound which cannot be improved by state transformations . Olas in [16] used piecewise quadratic functions to improve the bounds in the case of structured perturbations . A good review of the field is given in [4] .

1.2 Motivation :

Most of the dynamic systems are inherently nonlinear in nature . Unfortunately , solutions of majority of the nonlinear differential equations are not elementary functions and may be obtained only in a form of infinite series , coefficient for each term to be found independently . This leaves a person with a choice of only numerical solution , which again requires uniqueness and existence of a solution . If this condition is not met , even numerical integration would not give any realistic or dependable solution .

On the other hand , investigation of a nonlinear system by linearizing the equations and predicting the behaviour of the nonlinear system on the basis of the linearized equations might not often be very accurate in cases like input-output stability , trajectory control , etc. The analysis of stability robustness of a linear time-invariant system subject to perturbations (parameter variations) has been of considerable interest to researchers

types of perturbations , namely , time varying and time invariant perturbations .The selection of the type clearly influence the analysis . The stability of a linear time invariant system in the presence of time invariant perturbations (Hurwitz invariance , [13]) is basically addressed by testing for the negativity of the real parts of the eigenvalues (either in frequency domain or time-domain treatments), whereas the time-varying case is known to be best handled by the time-domain Liapunov stability analysis.

A study of stability robustness conditions of a dynamic system is very important from several practical points of interest .

A greater region for stability would imply better controller specifications and performance .

Also , a more robust system would be less sensitive to external disturbance and parameter uncertainties.

Finally , known upper bounds of perturbations which may be related with system parameters would allow a designer choose values for the parameters while keeping the system stable about the zero equilibrium state .

Recent widespread interest in robust design of control systems subject to structured (parametric) and unstructured perturbations shifted a considerable part of the research activity toward parameter space methods and , at the same time , enlarged the scope of the approach to include the Liapunov method as well as frequency domain concepts .

Compared to any other existing method , the obvious advantage of the Liapunov type techniques is that they easily can accomodate nonlinear time-varying and stochastic elements in perturbations . Also , the recent advancement in microprocessor architecture gives rise to the possibility of using more complex control algorithm than the simpler usual PID type .

1.3 Engineering Applications :

Examination of the stability of a given system is a basic step in system analysis . If a system is disturbed in any manner at any given time , we would like to determine the effect of the disturbance on subsequent output to facilitate an efficient design of a state-feedback controller . If a system is initially in a state of equilibrium , than it will remain theoretically in that state thereafter . Liapunov stability theorems are concerned with trajectories of a system when the initial state is near to equilibrium . From an engineering point of view , this is of utmost importance since disturbances (e.g. , noise , uncertainties , component error etc.) are always present in any real system . When systems are disturbed by their surroundings , it is extremely difficult to come up with a mathematical model that would represent the actual

disturbance . It is more difficult to find a closed form solution for the system, with or without disturbance , even when in some cases disturbances can be modelled analytically . So , from the practical point of view , it is of great importance to obtain measures that define and find allowable perturbation bounds so that the stability of the original system may be maintained .

For reasons stated above , analysis and design of stability robustness of linear,time-invariant systems subject to nonlinear, time-varying perturbations had been a matter of considerable research interest for a number of years . Conventional design of feedback controllers require knowledge of the bounds for disturbances . The knowledge of disturbance bounds is also important in adaptive control and control of nonlinear & uncertain systems . Singh and Coelho in [17] uses ultimate boundedness while considering application to control of aircraft . Spong in [19] uses knowledge of bounds in the design of "robust outer loop" to achieve robust tracking in manipulator control . Wide range of applications can be found in the field of automation , aircraft control , automobile , manipulators etc .

General stability definitions in the sense of Liapunov and Liapunov theorems are presented in the following chapter. Then we discuss the numerical approach with the

(piecewise) recursive Liapunov functions and its development. The examples considered are discussed along with the development of associative system .

In the concluding chapter , Liapunov based approaches for the determination of stability conditions of nominally (stable) linear dynamic systems with nonlinear timevarying unstructured perturbations are summarized with indications for future studies. Equations and source codes are included in the appendices.

2. DEFINITIONS AND CONCEPTS

2.1 Stability In The Sense Of Liapunov :

The three basic concepts of Liapunov theory are stability , asymptotic stability and global asymptotic stability . Roughly speaking , stability in the sense of Liapunov is related to the system trajectories depending continuously on the initial state ; asymptotic stability corresponds to trajectories that start sufficiently close to an equilibrium point actually converging to the equilibrium state as $t \rightarrow \infty$; and global asymptotic stability corresponds to every trajectory approaching a unique equilibrium point as $t \rightarrow \infty$.

Let us consider the vector differential equation

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}\,[t, \boldsymbol{x}(t)] \quad , \qquad t \geq 0 \qquad \text{------- (1)}$$

Throughout the discussion , we shall assume that the function $\boldsymbol{f}$ is of such a nature that the equation (1) has a unique solution over $[0,\infty)$ corresponding to each initial condition for $\boldsymbol{x}(0)$ and that this solution depends continuously on $\boldsymbol{x}(0)$.

We shall also assume that the vector $\boldsymbol{0}$ is an equilibrium point of the system (1) . This assumption does not result in any loss of generality [Vidyasagar] .

(a) Definition of stability in the sense of Liapunov :

The equilibrium point **0** at time t_0 of (1) is said to be *stable* at time t_0 if , for each $\varepsilon > 0$ there exists a $\delta(t_0, \varepsilon) > 0$ such that

$$\|x(t_0)\| < \delta(t_0, \epsilon) \quad \Rightarrow \quad \|x(t)\| < \epsilon \ , \ \forall\, t \geq t_0$$

It is said to be *uniformly stable* over $[t_0, \infty)$ if , for each $\varepsilon > 0$ there exists a $\delta(\varepsilon) > 0$ such that

$$\|x(t_1)\| < \delta(\epsilon) \quad , \quad t_1 \geq t_0 \Rightarrow \|x(t)\| < \epsilon \ , \ \forall\, t \geq t_1$$

[i.e., the same $\delta(\varepsilon)$ applies for all t_1]

(b) Definition of instability in the sense of Liapunov :

The equilibrium point **0** at time t_0 is unstable if it is not stable at time t_0 .

(c) Definition of asymptotic stability in the sense of Liapunov:

The equilibrium point **0** at time t_0 is asymptotically stable at time t_0 if (1) it is stable at time t_0 , and (2) there exists a number $\delta_1(t_0) > 0$ such that

$$\|x(t_0)\| < \delta_1(t_0) \quad \Rightarrow \|x(t)\| \to 0 \ \ as \ \ t \to \infty$$

It is *uniformly asymptotically* stable over $[t_0, \infty)$ if (1) it is uniformly stable over $[t_0, \infty)$, and (2) there

exists a number $\delta_1 > 0$ such that

$$\|\mathbf{x}(t_1)\| < \delta_1 \quad , \quad t_1 \geq t_0 \quad \Rightarrow \|\mathbf{x}(t)\| \to 0 \qquad as \quad t \to \infty$$

Moreover , the convergence is uniform with respect to t_1 .

(d) Definition of global asymptotic stability in the sense of Liapunov:

The equilibrium point **0** at time t_0 is *globally asymptotically stable* if $\mathbf{x}$ (t) $\to$ **0** as t $\to \infty$ [regardless of what $\mathbf{x}$ (t_0) is].

Some authors assume $t_0 = 0$, on the rationale that if t_0 is not equal to 0 , one can always "translate" the time variable .

A.M.Liapunov [1892] defines the solution of $\mathbf{x} = \mathbf{0}$ of (1) as asymptotically stable if it is stable and attractive .

We note here that stability , asymptotic stability and instability are basically local concepts , dealing with the trajectories of the system in the vicinity of an equilibrium point , whereas global asymptotic stability , as the name implies , is a global concept , having to do with the behaviour of all the trajectories of the system .

In his paper of 1892 , A.M.Liapunov proposed a fairly general definition for the stability of a solution of a differential equation. Roughly speaking , he said that a

solution $x^*(t)$ is stable with respect to some function $h(x)$, if the norm $\| h(x(t)) - h(x^*(t)) \|$ remains small whenever $\| x(t_0) - x^*(t_0) \|$ is chosen small enough. Later, several authors studied and extended his proposition.

Rouche in *Stability Theory by Liapunov's Direct Method* describes several other concepts of stability conditions also.

The solution $\mathbf{x} = \mathbf{0}$ of (1) is called *equi-asymptotically stable* (J.L.Massera) if it is stable and equi-attractive ; *uniformly asymptotically stable* (I.G.Malkin [1954]) if it is uniformly stable and uniformly attractive ; *globally asymptotically stable* (E.A.Barbashin and N.N.Krasovski [1952]) if it is uniformly stable and uniformly globally attractive.

In the case of autonomous and periodic systems , stability is equivalent to uniform stability and asymptotic stability is equivalent to uniform asymptotic stability [Vidyasagar].

Before we state Liapunov stability theorems , we present the concept of positive definite functions as well as the notion of the derivative along a trajectory.

(e) Definition of positive definite functions and locally positive definite functions :

A continuous function function $V : R_+ \times R^n \rightarrow R$ is

said to be locally positive definite function (l.p.d.f.) if there exists a continuous nondecreasing function $\alpha : R \rightarrow R$ such that

(i) $\alpha(0) = 0$, $\alpha(p) > 0$ whenever $p > 0$

(ii) $V(t,0) = 0$, $\forall\ t \geq 0$

(iii) $V(t,\mathbf{x}) \geq \alpha (\|\mathbf{x}\|)$, $\forall\ t \geq 0$, and for all $\mathbf{x}$ belonging to some ball $B_r = \{ \mathbf{x} : \|\mathbf{x}\| \leq r \}$, $r > 0$

V is said to be positive definite function (p.d.f.) if (iii) holds for all $\mathbf{x} \in R^n$ and in addition $\alpha(p) \rightarrow \infty$ as $p \rightarrow \infty$.

The type of function α considered above is said to belong to class K .

(f) Definition of decrescent functions :

A continuous function $V : R_+ \times R^n \rightarrow R$ is said to be decrescent if there exists a function $\beta(\cdot)$ belonging to class K such that $V(t,\mathbf{x}) \leq \beta(\|\mathbf{x}\|)$, $\forall\ t \geq 0$, $\forall\ \mathbf{x} \in B_r$, $r > 0$ where B_r is a ball in R^n .

(g) Definition of derivative along a trajectory :

Let $V : R_+ \times R^n \rightarrow R$ be continuously differentiable with respect to all it's arguments , and let ∇V denote the gradient of V with respect to $\mathbf{x}$. Then $\dot{V}$: $R_+ \times R^n \rightarrow R$ is defined by

$$\dot{V}(t,\boldsymbol{x}) = \frac{\partial V}{\partial t}(t,\boldsymbol{x}) + \nabla V(t,\boldsymbol{x})\,\boldsymbol{f}(t,\boldsymbol{x})$$

and is called the derivative of V along the trajectories of (1).

2.2 Liapunov's Direct Method

We present the three basic kinds of theorems stemming from the direct method of Liapunov , namely : stability theorem , asymptotic stability theorems , and instability theorem .

Stability Theorem

Theorem 1

The equilibrium point **0** at time t_0 of (1) is stable if there exists a continously differentiable l.p.d.f V such that

$$\dot{V}(t,\boldsymbol{x}) \leq 0 \quad \forall\ t \geq t_0,\ \forall\ x \in B_r, \quad \textit{for some ball } B_r$$

Asymptotic Stability Theorems

Theorem 2

The equilibrium point **0** at time t_o of (1) is uniformly asymptotically stable over the interval $[t_o, \infty)$ if there exists a continuously differentiable decrescent l.p.d.f V such that $-\dot{V}$ is an l.p.d.f.

Theorem 3

The equilibrium point **0** at time t_0 of (1) is globally asymptotically stable if there exists a contiunously differentiable decrescent p.d.f V such that

$$\dot{V}(t, \boldsymbol{x}) \leq -\gamma(\| \boldsymbol{x} \|), \quad \forall\, t \geq t_o, \ \forall\, \boldsymbol{x} \in R^n$$

where gamma is a function belonging to class K.

Instability Theorem

Theorem 4

The equilibrium point **0** at time t_0 of (1) is unstable if there exists a continuously differentiable decrescent function $V : R_+ \times R^n \rightarrow R$ such that (i) $\dot{V}$ is an l.p.d.f. and (ii) $V(t, \mathbf{0}) = 0$, and there exist points $\mathbf{x}$ arbitrarily close to **0** such that $V(t_0, \mathbf{x}) > 0$.

We conclude discussion on basic concepts of stability and theorems on stability here . It's worth mentioning that the choice of a suitable Liapunov function is often a difficult task and the common practice is to choose some form of quadratic function to start with . The Liapunov conditions for stability are only sufficient (not necessary) . Therefore , if a particular Liapunov function does not give satisfactory answer to a problem , it's not possible to draw any conclusion from that . Instead , the designer has to restart analysis with a new choice of Liapunov function candidate .

Researchers are still trying to come up with recipes for best Liapunov function for different situations. Till now , it depends mostly on the experience and intuition of the analyst to choose a Liapunov function candidate which will suite best to his problem .

In the next chapter , we propose a relatively new type of Liapunov function candidate , first introduced in [10] . The properties and possible applications were indicated in [7] . We will utilize this concept of Liapunov function candidate to solve our problem .

3. LIAPUNOV FUNCTION CANDIDATE

3.1 Integral Liapunov Functions :

The classic efforts to construct a Liapunov function for a given system are well reviewed by Hahn [8]. In this chapter , we discuss the procedure of designing recursive Liapunov functions for autonomous asymptotically stable systems . The concept and properties of recursive Liapunov functions were introduced in [10] and [7] .

Now, let us consider a linear stationary system

$$\dot{x} = Ax \qquad x \in R^n \tag{A}$$

with the solution $p(t,x_0)$, $p(0,x_0) = x_0$.

Let us consider the quadratic Liapunov function

$$V = x^T P x$$

where P is a symmetric positive definite matrix . We get ,

$$\dot{V}(x) = x^T(A^T P + PA)x$$

This is the basis of a traditional approach for the determination of stability status of a multi-dimensional linear dynamic system .

In [10] the recursive algorithm for design of

Liapunov function was introduced by defining the sequence of functions

$$V_{i+1}(x) = \int_0^T V_i\,(p(t,x))\,dt \qquad , \quad i=1,2,\ldots.$$

(B)

where T > 0 is some constant .

It was proved that the Liapunov derivative of $V_i(x)$ was given by the formula

$$\dot{V}_{i+1}(x) = \int_0^T \dot{V}_i(p(t,x))\,dt = V_i(p(T,x)) - v_i(x) \qquad , \quad i=1,2,.$$

Similarly the second Liapunov derivative of $V_i(x)$ may be expressed as

$$\ddot{V}_{i+1}(x) = \int_0^T \ddot{V}_i(p(t,x))\,dt = \dot{V}_i(p(T,x)) - \dot{V}_i(x) \qquad , \quad i=1,2,$$

The formula (B) will be used to generate an integral Liapunov function .

4. ALGORITHM AND EXAMPLES

4.1 Introduction :

We repeat here the problem statement before describing the algorithm . The robust stability problem for nominally linear system subject to nonlinear , time-varying , unstructured perturbation is considered . The system is of the form

$$\dot{x} = A_N x + e \tag{1}$$

where $x \in R^n$ and $e = e(t,x)$, $e(t,0) = 0$, is an unstructured perturbation . We define the problem as the assignment to determine the largest number $k > 0$ such that if $\|e\| < k\|x\|$ then the trivial solution to the system (1) is stable in the sense of Liapunov .

To accomodate nonlinear , time-varying perturbation $e(t,x)$ we utilize the Liapunov direct method to determine the unstructured perturbation bound k .

The selection and design of Liapunov function candidate was described in the previous chapter . The Liapunov function candidate is generated by means of the algorithm

$$V_{i+1}(x) = \int_0^T V_i(p(t,x))\,dt , \qquad i=1,2,3. \tag{2}$$

where p(t,x) , p(0,x) = x , is a solution to the system and T is an intgration interval of a finite length .

4.2 Algorithm :

A direct application of the algorithm (2) to the system (1) is impossible due to the fact that for any initial condition more than one solutions may exist depending on the shape of perturbation . To overcome this difficulty ,in the present paper, the original concept of the Liapunov function candidate is modified . Instead of the solution p(t,x) , the family of functions $\pi(t,x)$, continuously dependent on the parameter x , is considered .

$$V_{i+1}(x) = \int_0^T V_i\,(\pi(t,x))\,dt \tag{3}$$

We consider the function V_1 - the generator of the algorithm - to be of quadratic form , $V_1\,(x) = x^TSx$, where S is the solution of the Liapunov equation

$$A^TS + SA = -I \tag{4}$$

with I being an identity matrix .

The derivative of V_1 (x) along the solutions of (1) is of the form

$$\dot{V}(x) = -x^T x + 2x^T S e \tag{5}$$

Consequently , a continuous perturbation $e = e^*$, bounded by k and such that it maximizes the derivative at each point $x \in R^n$ is of the form

$$e^*(x,t) = \begin{cases} k\|x\| \dfrac{Sx}{\|Sx\|} & \text{for } \|x\| \neq 0 \\ 0 & \text{for } \|x\| = 0 \end{cases} \tag{6}$$

This form of perturbation is used to define an associated system of (1) given by the following equation

$$\dot{x} = A_N x + e^* \tag{7}$$

A general solution to this associated system continuously depends on its initial condition **x** . We denote this solution as the family π (t,x) .

Calculation of the derivative :

Let π (t,x) be the solution of the system described by equation (7) with π (0,x) = x . We generate

the Liapunov function by the formula

$$V_{i+1}(x) = \int_0^T V_i(\pi(\tau, x))\, d\tau \tag{8}$$

We observe that π (t,x) is not a solution to (1) . To obtain the formula for the Liapunov derivative we first express V as a function of t

$$V_{i+1}(p(t,x)) = \int_0^T V_i(\pi(\tau, p(t,x)))\, d\tau \tag{9}$$

Finally we have to differentiate with respect to t and put t = 0 . We have

$$\dot{V}_{(i+1)|t=0} = \int_0^T \sum_{i=j}^{n} \left[\frac{\partial V}{\partial \pi_i}(\tau, x) \left(\sum_{j=1}^{n} \frac{\partial \pi_i}{\partial p_j}(\tau, x)\ F_j(x)\right)\right] d\tau \tag{10}$$

where $\partial V/\partial \pi_i$ and $\partial \pi_i/\partial p_j$ are calculated for arguments τ and x , and F_j for the arguments x as marked .

Variational Equation :

Reference : Kurzweil j., Ordinary Differential

Equations , Elsevier , 1986 , pp 230-232 . The equation

$$\begin{bmatrix} \frac{d}{dt}\left(\frac{\partial \pi_1}{\partial p_i}\right) \\ \frac{d}{dt}\left(\frac{\partial \pi_2}{\partial p_i}\right) \\ \vdots \\ \frac{d}{dt}\left(\frac{\partial \pi_n}{\partial p_i}\right) \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_n} \end{bmatrix} \begin{bmatrix} \frac{\partial \pi_1}{\partial p_i} \\ \frac{\partial \pi_2}{\partial p_i} \\ \vdots \\ \frac{\partial \pi_n}{\partial p_i} \end{bmatrix} \tag{11}$$

describes how the i-th vector function col$[\partial\pi_1/\partial p_i$ $,\ldots\ldots\ldots,\partial\pi_n/\partial p_i]$ varies with time . The i-th vector function is the solution to the above equation with the initial condition $\partial\pi_1/\partial p_i = 0$ $,\ldots\ldots,\partial\pi_i/\partial p_i = 1,\ldots\ldots,$ $\partial\pi_n/\partial p_i = 0$. There are n such systems of equations , all together they form a matrix equation .

We use the solution to this matrix equation to generate the matrix $\partial\pi_i/\partial p_j$ .The knowledge of this matrix makes possible the determination of derivative of V along the solution trajectory of (1) utilizing equation (10) .

We considered two examples to compare results and to show how the algorithm works . We shall describe the examples now with descriptions of steps necessary for numerical simulation . The equations developed for the examples are listed in appendices I and II . The source codes for the simualations are listed in appendix III .

4.3 Example One :

Let us consider the two dimensional nominally linear dynamic system described by the equation

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -3 & -2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \tag{12}$$

To find the matrix S , we solve the Liapunov equation (4) and obtain

$$S = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 1.25 \end{bmatrix}$$

Equation (7) now describes the associated system with $e^*(x,t)$ given by (6) . Details are listed in appendix I.

Design Procedure :

step 1 :

Consider points on a unit circle to represent all possible initial conditions for the two-dimensional system . Points on a unit circle are described by

$$x_1 = \cos(\theta_1)$$
$$x_2 = \sin(\theta_1)$$

step 2 :

Begin simulation with $x_1(t=0) = 1.0$ and $x_2(t=0) = 0.0$, that is , for $\theta = 0.0$

step 3 :

Select a value of k and find the right hand side of the system (i.e. $F_j(x)$) at time t=0 with these initial conditions .

step 4 :

Use the two by two identity matrix as the initial condition for the variational equation .

step 5 :

Find $\partial V/\partial \pi_i$ for time t=0.0 .

step 6 :

Using informations from step 2 through 5 , calculate the derivative of the Liapunaov function candidate V at time t=0.0 .

step 7 :

Solve the set of differential equation describing the associated system using numerical methods ; get $\pi_i(t=t_1)$ where t_1 represents a small increment of time .

step 8 :

Find $\partial V/\partial \pi_i$ for time $t=t_1$.

step 9 :

Solve the set of differential equation describing the variational equations using numerical methods ; get

$\partial\pi_i/\partial p_j$ (at $t=t_1$) where t_1 represents same small increment of time .

step 10 :

Using informations from steps 7 through 9 and from step 3 , calculate the derivative of the Liapunov function candidate V at time $t=t_1$.

step 11 :

Add this value with the value obtained from step 6 .

step 12 :

Repeat steps 7 through 11 keeping the increment of time constant .

step 13 :

If the final value of the derivative of V is less than 0 then the system is stable for this value of initial condition (step 2) and k (step 3) . If the derivative of V turns out to be positive , then the procedure from step 2 through step 12 has to be repeated with a lower value of the parameter k. Otherwise , we select the next possible initial condition from the unit circle by incrementing angle θ by one degree . All points are covered like this till a value of the parameter k gives negative $\dot{V}$ corresponding to all possible initial conditions described by these points .

step 14 :

Next step is to find the maximum value of k by repeating the procedure described above by incrementing k till it gives positive values for $\dot{V}$.

4.4 Example Two :

Here we consider a three dimensional case described by the nominal system matrix

$$A_N = \begin{bmatrix} -2 & 0 & -3 \\ 0 & -3 & 0 \\ -1 & -1 & -4 \end{bmatrix}$$

Solving the Liapunov equation

$$A^T S + SA = -I$$

we get ,

$$S = \begin{bmatrix} 0.5714 & 0.0380 & -.1429 \\ 0.0380 & 0.3482 & -.0462 \\ -.1429 & -.0462 & 0.2857 \end{bmatrix}$$

As before , the system is given by ,

$$\dot{x} = A_N x + e$$

where $x \in R^n$ and $e = e(t,x)$, $e(t,0) = 0$, is an unstructured perturbation .

We know that ,a continuous perturbation $e = e^*$, bounded by k and such that it maximizes the derivative at each point $x \in R^n$ is of the form

$$e^*(x,t) = \begin{cases} k\|x\| \dfrac{Sx}{\|Sx\|} & \text{for } \|x\| \neq 0 \\ 0 & \text{for } \|x\| = 0 \end{cases}$$

So , this form of perturbation is used to define an associated system of (1) given by the following equation

$$\dot{x} = A_N x + e^*$$

The procedure for simulating the system to give an upper bound 'k' is same as the two dimensional system , except the fact that we need to use the points on a unit sphere instead of the unit circle of the two dimensional case.

The unit sphere is described by the equations

$$x_1 = \cos(\theta_1)\cos(\theta_2)$$
$$x_2 = \sin(\theta_1)\cos(\theta_2)$$
$$x_3 = \sin(\theta_2)$$

Varying θ_1 and θ_2 we can obtain different point on the unit sphere . The rest of the procedure is same .

Appendix II contains some equations used to obtain differnt expressions in the simulation process .

5. RESULTS AND CONCLUSIONS

5.1 Result Of Example One :

We described the system as $\dot{x} = A_x x + e$ where $x \in R^n$ and $e = e(t,x)$, $e(t,0) = 0$, is an unstructured perturbation . We defined the problem as the assignment to determine the largest number $k > 0$ such that if $|e| < k|x|$ then the trivial solution to the aforementioned system is stable in the sense of Liapunov .

We present here the simulation result along with the results obtained from two other algorithms :

Yedavalli and Liang	:	k = 0.394
Becker and Grimm	:	k = 0.540
Integral Liapunov Function	:	k = 0.493

5.2 Result of Example Two :

We present here the simulation result along with the result obtained from traditional Liapunov function approach :

Eigenvalue method	:	k = 0.781
Integral Liapunov Function	:	k = 1.500

5.3 Conclusions :

The results obtained via Integral Liapunov Functions are better than the traditional matrix norm approach .

The Integral Liapunov Function approach gives improved bound estimation than that of the state transformation method . Therefore further search for integral Liapunov functions is feasible . Improvement of the estimate is not large enough to exceed the estimate obtained by the small gain theorem (Becker and Grimm) .

The Integral Liapunov Function approach at the present form is not feasible for large systems , because execution time for computer simulation becomes very large with the increase in system dimension and numerical accuracy . For two or three dimensional nominally linear dynamic systems this can be easily implemented with a reasonable degree of accuracy.

5.4 Future Study :

(1) The concept of an associate system leads to

estimates improvement . A search for simple associate systems with differentiable right hand sides should be continued .

(2) The main difficulty of the algorithm is the need to solve the variational equation . A search for approximation of this equation should be made .

(3) There is scope to improve the algorithm to make it more efficient . Alternative Liapunov Functions can be designed .

(4) The algorithm can be applied to structured perturbations as well . It's application to structured perturbations is expected to improve the "conservatism" in the time domain stability robustness bounds also .

(5) High requirement on computer memory and speed for higher dimensional systems is an obstacle which can only be overcome through an improvement on the overall design procedure of the Liapunov Function candidate .

BIBLIOGRAPHY

1. Brayton,R.K.,and Tong,C.H.,"Stability of dynamical systems : A constructive approach",IEEE Trans. Circ. Syst., vol.26, no.4,1979, pp 224-234.

2. Brayton,R.K., and Tong,C.H.,"Constructive stability and asymptotic stability of dynamic systems", IEEE Trans. Circ. Syst., vol.27, 1980, pp 1121-1130.

3. Vidyasagar,M.,"Nonlinear system analysis",Prentice-Hall Hall,1978.

4. Siljak,D.D.,"Parameter space methods for robust control design : a guided tour ",IEEE Trans. Automat. Contr. , vol.34 ,no.7, Jul 1989, pp 674-688.

5. Olas,A.,"Recursive lyapunov functions",Journal of Dyn. sys. and Contr., vol III , Dec. 1989, pp 641-625.

6. Leitmann, G. and Corless,M.J.,"Continuous state feedback guaranteeing uniform ultimate boundedness for uncertain dynamic systems",IEEE Trans. Autom. contr., vol. AC-26, No.5, Oct.1981 , pp 1139-1144.

7. Olas,A.,"Recursive lyapunov functions : Properties, Linear systems", Control and Dyn. Syst., vol.35,1990.

8. W.Hann,"Stability of motion",Springler-Verlag , New York , 1967.

9. K.Ogata,"Modern control engineering",Prentice-Hall , 1970.

10. Olas,A.,"Recursive lyapunov functions",Workshop on control mechanics,USC,January,1988.

11. B.Radziszewski,"On the best liapunov functions",IFTR Reports,Polish Academy Of Sciences,Warsaw,1977.

12. Becker,N. and Grimm,W.,"Comments on reduced conservatism in stability robustness bounds by state transformation",IEEE Trans. on Autom. Contr., vol.33 ,no.2,Feb 1988.

13. Yedavelli,R.K. and Liang,Z.,"Reduced conservatism in stability robustness bounds by state transformation ", IEEE Trans. on Auto. Contr.,vol.AC-31,no.9,Sept.1986.

14. D.Siljak,"Large scale dynamic systems : stability and structure",North-Holland , Amsterdam , The Netherlands , 1978.

15. R.V.Patel and M.Toda,"Quantitative measures of robustness of multivariable systems",Proc. JACC , SanFrancisco , TP8-A , 1980.

16. Olas,A.,"On robustness of systems with structured uncertainties",Proceedings of the IVth workshop on control mechanics, University of Southern California, January,1991,in print.

17. Narendra,K.S., and Tripathy,S.S. ," Identification and optimization of aircraft dynamics ", Journal Of Aircraft , vol. 10, no. 4, Apr. 1973, pp. 193-199 .

18. Singh,S.N., and Coehlo,A.A.R.,"Nonlinear control of mismatched uncertain linear systems and application to control of aircraft",Journal of Dynamic Systems , Measurement , and Control,vol. 106, Sep. 1984, pp.106-210.

19. Spong,Mark,W. and Vidyasagar,M. , " Robot Dynamics and Control " , Wiley , 1989 .

Appendices

APPENDIX I
Equations for example one

Here, we list few equations used in solving example one , the two dimensional nominally linear system . The system is of the form

$$\dot{x} = A_N x + e \qquad (1)$$

where A_N and **e** was defined before . Also , we chose Liapunove function candidate to be of the form $V = x^T S x$, the two by two matrix S was found by solving the Liapunov equation . Now we have ,

$$Sx = \frac{1}{4}\begin{bmatrix} x_1+x_2 \\ x_1+5x_2 \end{bmatrix}$$

$$\|x\| = \sqrt{x_1^2 + x_2^2}$$

$$\|Sx\| = \frac{1}{4}\sqrt{2x_1^2 + 12x_1x_2 + 26x_2^2}$$

$$V = x^T S x$$

Again ,

$$\begin{aligned} \dot{V} &= \dot{x}^T S x + x^T S \dot{x} \\ &= (x^T A^T + e^T) S x + x^T S (Ax+e) \\ &= x^T (A^T S + SA) x + e^T S x + x^T S e \\ &= - x^T x + 2 x^T S e \end{aligned}$$

Writing R.H.S. of (1) as F_j , we have for $\| x \| \neq 0$,

$$F_1 = -3x_1 - 2x_2 + k\frac{\sqrt{x_1^2+x_2^2}}{\sqrt{2x_1^2+12x_1x_2+26x_2^2}}(x_1+x_2)$$

$$F_2 = x_1 + k\frac{\sqrt{x_1^2+x_2^2}}{\sqrt{2x_1^2+12x_1x_2+26x_2^2}}(x_1+5x_2)$$

To get $\partial V/\partial \pi_i$, and denoting π_i as x_i , we have ,

$$\frac{\partial V}{\partial x_1} = \frac{1}{4}\frac{\partial}{\partial x_1}\left([x_1 \quad x_2]\begin{bmatrix}x_1+x_2\\ x_1+5x_2\end{bmatrix}\right)$$
$$= -\frac{1}{4}\frac{\partial}{\partial x_1}(x_1^2+2x_1x_2+5x_2^2)$$
$$= -\frac{1}{4}(2x_1+2x_2)$$
$$= -\frac{1}{2}(x_1+x_2)$$

$$\frac{\partial V}{\partial x_2} = -\frac{1}{4}\frac{\partial}{\partial x_2}(x_1^2+2x_1x_2+5x_2^2)$$
$$= -\frac{1}{4}(2x_1+10x_2)$$
$$= -\frac{1}{2}(x_1+5x_2)$$

To get $\partial \pi_i/\partial p_j$, we have to solve variational equation given by,

$$\frac{d}{dt}\left[\frac{\partial \pi_l}{\partial p_j}\right] = \sum_{r=1}^{r=n}\left(\frac{\partial F_l}{\partial \pi_r}\right)\left(\frac{\partial \pi_r}{\partial p_j}\right)$$

with the initial condition given by an nxn identity matrix . The elements of the matrix $\partial F_i/\partial \pi_r$ are given by , denoting π_r by x_r , and F_i defined as before , when $\| x \| \neq 0$,

$$\frac{\partial F_1}{\partial x_1} = -3 + \frac{k\|x\|}{4\|Sx\|} + \frac{kx_1(x_1+x_2)}{4\|Sx\|\,\|x\|} - \frac{2k\|x\|(x_1+3x_2)(x_1+x_2)}{\sqrt{(2x_1^2+12x_1x_2+26x_2^2)^3}}$$

$$\frac{\partial F_1}{\partial x_2} = -2 + \frac{k\|x\|}{4\|Sx\|} + \frac{kx_2(x_1+x_2)}{4\|Sx\|\,\|x\|} - \frac{2k\|x\|(3x_1+13x_2)(x_1+x_2)}{\sqrt{(2x_1^2+12x_1x_2+26x_2^2)^3}}$$

$$\frac{\partial F_2}{\partial x_1} = 1 + \frac{k\|x\|}{4\|Sx\|} + \frac{kx_1(x_1+5x_2)}{4\|Sx\|\,\|x\|} - \frac{2k\|x\|(x_1+3x_2)(x_1+5x_2)}{\sqrt{(2x_1^2+12x_1x_2+26x_2^2)^3}}$$

$$\frac{\partial F_2}{\partial x_2} = \frac{5k\|x\|}{4\|Sx\|} + \frac{kx_2(x_1+5x_2)}{4\|Sx\|\,\|x\|} - \frac{2k\|x\|(3x_1+13x_2)(x_1+5x_2)}{\sqrt{(2x_1^2+12x_1x_2+26x_2^2)^3}}$$

when $\| x \| = 0$,

$$\frac{\partial F_1}{\partial x_1} = -3 \quad , \quad \frac{\partial F_1}{\partial x_2} = -2$$

$$\frac{\partial F_2}{\partial x_1} = 1 \quad , \quad \frac{\partial F_2}{\partial x_2} = 0$$

APPENDIX II
Equations for example two

For systems with dimensions higher than two , the partial derivatives expressions for the varitional equations are best handled by indicial notations . The following equations were used in example two .

(a) When A_{ij} is a constant matrix , we have ,

$$y_i = A_{ij}x_j$$
$$\frac{\partial y_i}{\partial x_m} = A_{ij}\delta_{jm} = A_{im}$$

(b) Representing x_ix_i as a sum with i=1,2,3

$$\|\mathbf{x}\| = (x_ix_i)^{1/2}$$
$$\frac{\partial \|\mathbf{x}\|}{\partial x_m} = \frac{1}{2}(x_ix_i)^{-1/2}(2x_i\delta_{im}) = \frac{x_m}{\|\mathbf{x}\|}$$

(c) Let , $\mathbf{Sx} = y_i$, then ,

$$\|\mathbf{Sx}\| = (y_iy_i)^{1/2}$$

and

$$\frac{1}{\|\mathbf{Sx}\|} = (y_iy_i)^{-1/2}$$

then

$$\frac{\partial}{\partial x_m}\left(\frac{1}{\|\mathbf{Sx}\|}\right) = -\frac{1}{2}(y_iy_i)^{-3/2}(2y_iS_{im})$$
$$= -(y_iy_i)^{-3/2}y_iS_{im}$$
$$= \frac{-y_iS_{im}}{(y_iy_i)^{3/2}}$$

APPENDIX III
Source Codes

In this section , we present the main files of the source codes to simulate the three dimensional dynamic system (example two) .

The source codes are written in 'C' . They are presented without comments , primarily in an effort to keep the volume concise . The header files along with some utility files are omitted also .

The source codes can be used to determine the bounds of systems of arbitrary order with a few changes (e.g. , the " #define n 3 " to "#define n ...the dimension here.. " , matrices A_{ij} , S_{ij} etc) . However, dimension higher than 3 would make the execution time very high . Changes in 'time steps' , 'total time for integration' etc. could make it feasible for high-speed computing machines . The choice of indicial notations was primarily made to accomodate changes in system dimension easily without going through much of the algebra one has to do otherwise . More efficient programs can be easily written if one is restricted to a fixed system dimension .
The requirement for high execution time was partially removed by working with larger grids for search (large increments in angles to move to a new point on unit sphere) initially to have a rough estimate on the bound and finally making a single run with increments of one degree for $theta_1$ and $theta_2$ to confirm the rough estimate .

The files rk4.c , rkqc.c and odeint.c are taken from " Numerical Recipes ,The Art of Scientific Computing " by Press,W.H.,Flannery,B.P.,Tenkolosky,S.A. and Vetterling,W.T. The programs were run with other algorithms for solving a set of ordinary differential equations (namely , predictor-corrector scheme ,fifth order Runge-Kutta ,Bulirsch_Stoer method ,etc.) also to see if the proposed algorithm is sensitive to a choice of a particular scheme . It was found that the proposed algorithm is not dependent on the choice of

a particular scheme .

The file "matutil.c" was written to facilitate certain matrix and vector operations . The file "tstn3D.c" contains the main functions and the file "tstnsp3.c" contains some supportive functions for the main .

```
                    /*  File  :  tstn3D.c  */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "fix1.h"
#include "fix2.h"
#define PI acos(-1.0)
#define n 3
#define fstp PI/180.0
#define step 0.1
#define T 2.0
#define total 1
double delF_delpii(),delv_delpii();
double vdott();
double **s,**a,**ac;
double k,*fj;
double vdot;
void system(),process();
main()
{
int j,r,Numb;
double kmax,kmin;
void init(),finit();
double *xrecc,*xc,v1,v2,v3;
double limit1,limit2;
FILE *fopen(),*data;
data=fopen("tsis3D10.dat","w");
xrecc=dvector(1,n);
xc=dvector(1,n);
init();
kmax = 2.05;
kmin = 1.05;
Numb = 1;
limit1=PI+0.5*fstp;
limit2=2.0*PI+0.5*fstp;

do
{
k = 0.5*(kmax+kmin) ;
for ( v1 = 0.0 ; v1 <= limit1 ; v1+=fstp )
 {
   for ( v2 = 0.0 ; v2 <= limit2 ; v2+=fstp )
    {
      xc[1]=cos(v1)*cos(v2);
      xc[2]=sin(v1)*cos(v2);
      xc[3]=sin(v2);
      process(xc);
      if( vdot >= 0 ) break;
    }
```

```
    if ( vdot >= 0.0 )
     {
      kmax = k ;
      Numb +=1;
        fprintf(data,"k=%5.2lf    v1=%5.2lf    v2=%5.2lf
vdot=%5.2lf\n",k,v1*180.0/PI,v2*180.0/PI,vdot);
      break;
     }
  }
if ( vdot < 0.0 )
{
kmin = k ;
Numb = Numb+1 ;
fprintf(data,"k=%5.2lf    v1=%5.2lf    v2=%5.2lf
vdot=%5.2lf\n",k,(v1-fstp)*180.0/PI,(v2-fstp)*180.0/PI,vdot);
}
} while ( Numb <= total );

free_dvector(xrecc,1,n);
free_dvector(xc,1,n);
finit();
return;
}



void process(xx)
double xx[];
{
int e;
double *x,tbegin,*dxdt,tend,*newxrecc;
void integrate();
x=dvector(1,n+n*n);
dxdt=dvector(1,n+n*n);
newxrecc=dvector(1,n+n*n);
tbegin=0.0; tend=step;
for(e=1;e<=n;e++) x[e]=xx[e];
convmv(ac,x,(n+1));
system(tbegin,x,dxdt);
for(e=1;e<=n;e++) fj[e]=dxdt[e];
vdot=vdott(x);
do
{
integrate(x,tbegin,tend,newxrecc);
vdot+=vdott(newxrecc);
for(e=1;e<=n+n*n;e++) x[e]=newxrecc[e];
tbegin=tend; tend+=step;
} while(tend<=T);
free_dvector(x,1,n+n*n);
free_dvector(dxdt,1,n+n*n);
free_dvector(newxrecc,1,n+n*n);
return;
}

extern double dxsav,*xp,**yp;
extern int kount,kmax;

void integrate(ystart,x1,x2,ynd)
double ystart[],x1,x2,ynd[];
{
```

```
int  i,nbad,nok;
double eps,h1,hmin;
void rkqc(),rk4(),odeint();
xp=dvector(1,200);
yp=dmatrix(1,n+n*n,1,200);
eps=1.0e-4;
h1=(x2-x1)/2.0;
hmin=0.0;
kmax=100;
dxsav=(x2-x1)/1.0;
odeint(ystart,n+n*n,x1,x2,eps,h1,hmin,&nok,&nbad,system,rkqc);
for(i=1;i<=n+n*n;i++)
 ynd[i]=yp[i][kount];
free_dmatrix(yp,1,n+n*n,1,200);
free_dvector(xp,1,200);
}
#undef n
#undef step
#undef total
#undef T
#undef PI
```

```
                    /*  File  :  tstnsp3.c   */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "fix1.h"
#include "fix2.h"
#define n 3

double delF_delpii(),delv_delpii();
double vdott();
extern double **s,**a,**ac;
extern double k,*fj;
extern double vdot;

double delF_delpii(pi,pj,xx,uu)
int pi,pj;
double xx[],uu[];
{
int i;
double dotvv(),dotus,sum,*temp;
double modv(),modxx,moduu;
temp=dvector(1,n);
for(i=1;i<=n;i++)
temp[i]=s[i][pj];
dotus=dotvv(uu,temp);
sum=xx[pj]*uu[pi];
modxx=modv(xx);moduu=modv(uu);
sum/=(modxx*moduu);
sum+=s[pi][pj]*modxx/moduu;
sum-=modxx*uu[pi]*dotus/(pow(moduu,3.0));
sum*=k;
sum+=a[pi][pj];
free_dvector(temp,1,n);
return sum;
}
```

```
double delv_delpii(m,xx)
int m;
double xx[];
{
int i;
double *temp1,*temp2,*temp3,simxi,smjxj;
double dotvv();
temp1=dvector(1,n);
temp2=dvector(1,n);
temp3=dvector(1,n);
for(i=1;i<=n;i++) temp3[i]=xx[i];
for(i=1;i<=n;i++)
temp1[i]=s[i][m];
for(i=1;i<=n;i++)
temp2[i]=s[m][i];
simxi=dotvv(temp1,temp3);
smjxj=dotvv(temp2,temp3);
free_dvector(temp1,1,n);
free_dvector(temp2,1,n);
free_dvector(temp3,1,n);
return (simxi+smjxj);
}


void init()
{
int i,j;
ac=dmatrix(1,n,1,n);
s=dmatrix(1,n,1,n);
a=dmatrix(1,n,1,n);
fj=dvector(1,n);
a[1][1]=-2.0;a[1][2]=0.0;a[1][3]=-1.0;
a[2][1]=0.0;a[2][2]=-3.0;a[2][3]=0.0;
a[3][1]=-1.0;a[3][2]=-1.0;a[3][3]=-4.0;
s[1][1]=0.5714;s[1][2]=0.0380;s[1][3]=-0.1429;
s[2][1]=0.0380;s[2][2]=0.3482;s[2][3]=-0.0462;
s[3][1]=-0.1429;s[3][2]=-0.0462;s[3][3]=0.2857;
for(i=1;i<=n;i++)
{
 for(j=1;j<=n;j++)
 {
  if(i==j) ac[i][j]=1.0;
   else ac[i][j]=0.0;
}}
return;
}

double vdott(svv)
double svv[];
{
int i,j,r;
double vd,**tmp1,*tmp2,*dvdpi;
tmp1=dmatrix(1,n,1,n);
tmp2=dvector(1,n);
dvdpi=dvector(1,n);
convvm(tmp1,svv,n+1);
mmv(tmp1,fj,tmp2);
for(i=1;i<=n;i++)
dvdpi[i]=delv_delpii(i,svv);
vd=dotvv(dvdpi,tmp2);
```

```
free_dmatrix(tmp1,1,n,1,n);
free_dvector(tmp2,1,n);
free_dvector(dvdpi,1,n);
return vd;
}

void system(x,y,dydx)
double x,y[],dydx[];
{
int i,j,r;
double *y1,*u1,*u2,*ax1,*xdmy,**gg,**hh;
double conn1,mody1,modu1;
  y1=dvector(1,n);
  u1=dvector(1,n);
  u2=dvector(1,n);
  ax1=dvector(1,n);
  xdmy=dvector(1,n);
  gg=dmatrix(1,n,1,n);
  hh=dmatrix(1,n,1,n);
for(i=1;i<=n;i++)
y1[i]=y[i];
convvm(gg,y,n+1);
mody1=modv(y1);
mmv(s,y1,u1);
modu1=modv(u1);
conn1=k*mody1/modu1;
mvc(u1,conn1,u2);
mmv(a,y1,ax1);
addvv(ax1,u2,xdmy);
for(i=1;i<=n;i++)
{
 for(j=1;j<=n;j++)
 {
  hh[i][j]=0.0;
  for(r=1;r<=n;r++)
  hh[i][j]+=delF_delpii(i,r,y1,u1)*gg[r][j];
 }
}
convmv(hh,dydx,n+1);
for(i=1;i<=n;i++)
dydx[i]=xdmy[i];
  free_dvector(y1,1,n);
  free_dvector(u1,1,n);
  free_dvector(u2,1,n);
  free_dvector(ax1,1,n);
  free_dvector(xdmy,1,n);
  free_dmatrix(gg,1,n,1,n);
  free_dmatrix(hh,1,n,1,n);
return;
}


void finit()
{
free_dmatrix(s,1,n,1,n);
free_dmatrix(a,1,n,1,n);
free_dvector(fj,1,n);
free_dmatrix(ac,1,n,1,n);
return;
}
```

```
#undef n


                        /*  File :  rk4.c   */
void rk4(y,dydx,n,x,h,yout,derivs)
double y[],dydx[],x,h,yout[];
void (*derivs)(); /* ANSI: void (*derivs)(float,float *,float *); */
int n;
{
      int i;
      double xh,hh,h6,*dym,*dyt,*yt,*dvector();
      void free_dvector();

      dym=dvector(1,n);
      dyt=dvector(1,n);
      yt=dvector(1,n);
      hh=h*0.5;
      h6=h/6.0;
      xh=x+hh;
      for (i=1;i<=n;i++) yt[i]=y[i]+hh*dydx[i];
      (*derivs)(xh,yt,dyt);
      for (i=1;i<=n;i++) yt[i]=y[i]+hh*dyt[i];
      (*derivs)(xh,yt,dym);
      for (i=1;i<=n;i++) {
            yt[i]=y[i]+h*dym[i];
            dym[i] += dyt[i];
      }
      (*derivs)(x+h,yt,dyt);
      for (i=1;i<=n;i++)
            yout[i]=y[i]+h6*(dydx[i]+dyt[i]+2.0*dym[i]);
      free_dvector(yt,1,n);
      free_dvector(dyt,1,n);
      free_dvector(dym,1,n);
}

                      /*   File :  rkqc.c    */
#include <math.h>

#define MAXSTP 10000
#define TINY 1.0e-30

int kmax=0,kount=0;  /* defining declaration */
double *xp=0,**yp=0,dxsav=0;  /* defining declaration */

void odeint(ystart,nvar,x1,x2,eps,h1,hmin,nok,nbad,derivs,rkqc)
double ystart[],x1,x2,eps,h1,hmin;
int nvar,*nok,*nbad;
void (*derivs)(); /* ANSI: void (*derivs)(float,float *,float *); */
void (*rkqc)();   /*  ANSI:  void  (*rkqc)(float  *,float  *,int,float
*,float,
                        float,float *,float *,float *,void (*)()); */
{
      int nstp,i;
      double xsav,x,hnext,hdid,h;
      double *yscal,*y,*dydx,*dvector();
      void nrerror(),free_dvector();
```

```
        yscal=dvector(1,nvar);
        y=dvector(1,nvar);
        dydx=dvector(1,nvar);
        x=x1;
        h=(x2 > x1) ? fabs(h1) : -fabs(h1);
        *nok = (*nbad) = kount = 0;
        for (i=1;i<=nvar;i++) y[i]=ystart[i];
        if (kmax > 0) xsav=x-dxsav*2.0;
        for (nstp=1;nstp<=MAXSTP;nstp++) {
              (*derivs)(x,y,dydx);
              for (i=1;i<=nvar;i++)
                    yscal[i]=fabs(y[i])+fabs(dydx[i]*h)+TINY;
              if (kmax > 0) {
                    if (fabs(x-xsav) > fabs(dxsav)) {
                          if (kount < kmax-1) {
                                xp[++kount]=x;
                                for (i=1;i<=nvar;i++) yp[i][kount]=y[i];
                                xsav=x;
                          }
                    }
              }
              if ((x+h-x2)*(x+h-x1) > 0.0) h=x2-x;
              (*rkqc)(y,dydx,nvar,&x,h,eps,yscal,&hdid,&hnext,derivs);
              if (hdid == h) ++(*nok); else ++(*nbad);
              if ((x-x2)*(x2-x1) >= 0.0) {
                    for (i=1;i<=nvar;i++) ystart[i]=y[i];
                    if (kmax) {
                          xp[++kount]=x;
                          for (i=1;i<=nvar;i++) yp[i][kount]=y[i];
                    }
                    free_dvector(dydx,1,nvar);
                    free_dvector(y,1,nvar);
                    free_dvector(yscal,1,nvar);
                    return;
              }
              if (fabs(hnext) <= hmin) nrerror("Step size too small in
ODEINT");
              h=hnext;
        }
        nrerror("Too many steps in routine ODEINT");
}

#undef MAXSTP
#undef TINY
```

/* File : odeint.c */

```
#include <math.h>

#define MAXSTP 10000
#define TINY 1.0e-30

int kmax=0,kount=0;  /* defining declaration */
double *xp=0,**yp=0,dxsav=0;  /* defining declaration */

void odeint(ystart,nvar,x1,x2,eps,h1,hmin,nok,nbad,derivs,rkqc)
double ystart[],x1,x2,eps,h1,hmin;
int nvar,*nok,*nbad;
void (*derivs)(); /* ANSI: void (*derivs)(float,float *,float *); */
void (*rkqc)();   /*  ANSI:  void  (*rkqc)(float  *,float  *,int,float
*,float,
```

```
                        float,float *,float *,float *,void (*)()); */
{
      int nstp,i;
      double xsav,x,hnext,hdid,h;
      double *yscal,*y,*dydx,*dvector();
      void nrerror(),free_dvector();

      yscal=dvector(1,nvar);
      y=dvector(1,nvar);
      dydx=dvector(1,nvar);
      x=x1;
      h=(x2 > x1) ? fabs(h1) : -fabs(h1);
      *nok = (*nbad) = kount = 0;
      for (i=1;i<=nvar;i++) y[i]=ystart[i];
      if (kmax > 0) xsav=x-dxsav*2.0;
      for (nstp=1;nstp<=MAXSTP;nstp++) {
            (*derivs)(x,y,dydx);
            for (i=1;i<=nvar;i++)
                  yscal[i]=fabs(y[i])+fabs(dydx[i]*h)+TINY;
            if (kmax > 0) {
                  if (fabs(x-xsav) > fabs(dxsav)) {
                        if (kount < kmax-1) {
                              xp[++kount]=x;
                              for (i=1;i<=nvar;i++) yp[i][kount]=y[i];
                              xsav=x;
                        }
                  }
            }
            if ((x+h-x2)*(x+h-x1) > 0.0) h=x2-x;
            (*rkqc)(y,dydx,nvar,&x,h,eps,yscal,&hdid,&hnext,derivs);
            if (hdid == h) ++(*nok); else ++(*nbad);
            if ((x-x2)*(x2-x1) >= 0.0) {
                  for (i=1;i<=nvar;i++) ystart[i]=y[i];
                  if (kmax) {
                        xp[++kount]=x;
                        for (i=1;i<=nvar;i++) yp[i][kount]=y[i];
                  }
                  free_dvector(dydx,1,nvar);
                  free_dvector(y,1,nvar);
                  free_dvector(yscal,1,nvar);
                  return;
            }
            if (fabs(hnext) <= hmin) nrerror("Step size too small in
ODEINT");
            h=hnext;
      }
      nrerror("Too many steps in routine ODEINT");
}

#undef MAXSTP
#undef TINY
```

```
                        /* File   matutil.c    */
#include <math.h>
#define N 3
```

```
void mmv(x,y,z)
double *x[],y[],z[];
{
int i,j;
for ( i=1 ; i<=N ; i++ )
 {
  z[i] = 0.0 ;
  for ( j=1 ; j<=N ; j++ )
   z[i] += x[i][j]*y[j];
 }
return;
}

void addvv(x,y,z)
double x[],y[],z[];
{
int i;
for ( i=1 ; i<=N ; i++ )
z[i] = x[i] + y[i] ;
return;
}

void addmm(x,y,z)
double *x[],*y[],*z[];
{
int i,j;
for ( i=1 ; i<=N ; i++ )
{
 for ( j=1 ; j<=N ; j++ )
  z[i][j]= x[i][j] + y[i][j];
}
return;
}

void mmc(x,c,y)
double *x[],c,*y[];
{
int i,j;
for ( i=1 ; i<=N ; i++ )
{
 for ( j=1 ; j<=N ; j++ )
  y[i][j]= c*x[i][j] ;
}
return;
}

void mvc(x,c,y)
double x[],c,y[];
{
int i;
for ( i=1 ; i<=N ; i++ )
y[i] = c*x[i] ;
return;
}

void convmv(x,y,g)
int g;
double *x[],y[];
{
int i,j,ki;
```

```
ki=g;
for ( i=1 ; i<=N ; i++ )
{
 for ( j=1 ; j<=N ; j++ )
  {
   y[ki]= x[i][j] ;
   ki += 1 ;
  }
}
return;
}

void convvm(x,y,g)
int g;
double *x[],y[];
{
int i,j,ki;
ki=g;
for ( i=1 ; i<=N ; i++ )
{
 for ( j=1 ; j<=N ; j++ )
  {
   x[i][j]=y[ki] ;
   ki += 1 ;
  }
}
return;
}

double dotvv(x,y)
double x[],y[];
{
int i;
double sum=0.0;
for ( i=1 ; i<=N ; i++ )
sum += x[i]*y[i] ;
return sum;
}

double modv(x)
double x[];
{
int i;
double sum=0.0;
for ( i=1 ; i<=N ; i++ )
sum += x[i]*x[i] ;
return sqrt(sum);
}

#undef N
```